TYPE Original Research
PUBLISHED 09 March 2023
DOI 10.3389/fncom.2023.1120516

OPEN ACCESS

EDITED BY
Jia-Bao Liu,
Anhui Jianzhu University, China

REVIEWED BY
Yinghao Chen,
Eastern Institute for Advanced Study, China
Zhengguang Liu,
Xi'an Jiaotong University, China

*CORRESPONDENCE
Hongli Sun
✉ honglisun@126.com

RECEIVED 10 December 2022
ACCEPTED 13 February 2023
PUBLISHED 09 March 2023

CITATION
Lu Y, Zhang S, Weng F and Sun H (2023) Approximate solutions to several classes of Volterra and Fredholm integral equations using the neural network algorithm based on the sine-cosine basis function and extreme learning machine.
Front. Comput. Neurosci. 17:1120516.
doi: 10.3389/fncom.2023.1120516

Approximate solutions to several classes of Volterra and Fredholm integral equations using the neural network algorithm based on the sine-cosine basis function and extreme learning machine

Yanfei Lu[1], Shiqing Zhang[1], Futian Weng[2] and Hongli Sun[3]*

[1]School of Electronics and Information Engineering, Taizhou University, Zhejiang, Taizhou, China, [2]Data Mining Research Center, Xiamen University, Fujian, Xiamen, China, [3]School of Mathematics and Statistics, Central South University, Hunan, Changsha, China

In this study, we investigate a new neural network method to solve Volterra and Fredholm integral equations based on the sine-cosine basis function and extreme learning machine (ELM) algorithm. Considering the ELM algorithm, sine-cosine basis functions, and several classes of integral equations, the improved model is designed. The novel neural network model consists of an input layer, a hidden layer, and an output layer, in which the hidden layer is eliminated by utilizing the sine-cosine basis function. Meanwhile, by using the characteristics of the ELM algorithm that the hidden layer biases and the input weights of the input and hidden layers are fully automatically implemented without iterative tuning, we can greatly reduce the model complexity and improve the calculation speed. Furthermore, the problem of finding network parameters is converted into solving a set of linear equations. One advantage of this method is that not only we can obtain good numerical solutions for the first- and second-kind Volterra integral equations but also we can obtain acceptable solutions for the first- and second-kind Fredholm integral equations and Volterra–Fredholm integral equations. Another advantage is that the improved algorithm provides the approximate solution of several kinds of linear integral equations in closed form (i.e., continuous and differentiable). Thus, we can obtain the solution at any point. Several numerical experiments are performed to solve various types of integral equations for illustrating the reliability and efficiency of the proposed method. Experimental results verify that the proposed method can achieve a very high accuracy and strong generalization ability.

KEYWORDS
Volterra-Fredholm integral equations, approximate solutions, neural network algorithm, sine-cosine basis function, extreme learning machine

1. Introduction

Volterra and Fredholm integral equations have many applications in natural sciences and engineering. A linear phenomenon appearing in many applications in scientific fields can be modeled by linear integral equations (Abdou, 2002; Isaacson and Kirby, 2011). For example, as mentioned by Lima and Buckwar (2015), a class of integro-differential equations, known as neural field equations, describes the large-scale dynamics of spatially structured networks of neurons. These equations are widely used in the field of neuroscience and robotics, and they also play a crucial role in cognitive robotics. The reason is that the architecture of autonomous robots, which are able to interact with other agents in dealing with a mutual task, is strongly inspired by the processing principles and the neuronal circuitry in the primate brain.

This study aims to consider several kinds of linear integral equations. The general form of linear integral equations is defined as follows:

$$\epsilon y(x)+\lambda \int_a^b k_1(x,t)y(t)dt+\mu \int_a^x k_2(x,t)y(t)dt = g(x), \qquad x \in [a,b], \tag{1}$$

Where the functions $k_1(x,t), k_2(x,t)$, and $g(x)$ are known, but $y(x)$ is the unknown function that will be determined; a and b are constants; and ϵ, λ and μ are parameters. Notably, we have

(i). Equation (1) is called linear Fredholm integral equation of the first kind if $\epsilon, \mu = 0$ and $\lambda = 1$.
(ii). Equation (1) is called linear Volterra integral equation of the first kind if $\epsilon, \lambda = 0$ and $\mu = 1$.
(iii). Equation (1) is called linear Fredholm integral equation of the second kind if $\mu = 0$ and $\epsilon, \lambda = 1$.
(iv). Equation (1) is called linear Volterra integral equation of the second kind if $\lambda = 0$ and $\epsilon, \mu = 1$.
(v). Equation (1) is called linear Volterra–Fredholm integral equation if $\mu, \epsilon, \lambda = 1$.

Many methods for numerical solutions of Volterra integral equations, Fredholm integral equations, and Volterra-Fredholm integral equations have been presented in recent years. Orthogonal polynomials (e.g., wavelets Maleknejad and Mirzaee, 2005, Bernstein Mandal and Bhattacharya, 2007, Chebyshev Dastjerdi and Ghaini, 2012) were proposed for solving integral equations. The Taylor collocation method (Wang and Wang, 2014), Lagrange collocation method (Wang and Wang, 2013; Nemati, 2015), and Fibonacci collocation method (Mirzaee and Hoseini, 2016) were effective and convenient for solving integral equations. The Sinc-collocation method (Rashidinia and Zarebnia, 2007) and Galerkin method (Saberi-Nadjafi et al., 2012) also give good performance in solving Volterra integral equation problems. However, most of these traditional methods have the following disadvantage: they provide the solution, in the form of an array, at specific preassigned mesh points in the domain, and they need an additional interpolation procedure to yield the solution for the whole domain. In order to have an accurate solution, one either has to increase the order of the method or decrease the step size. This, however, increases the computational cost.

The neural network has excellent application potential in many fields (Habib and Qureshi, 2022; Li and Ying, 2022) owing to its universal function approximation capabilities (Hou and Han, 2012; Hou et al., 2017, 2018). In this case, the neural network is widely used as an effective tool for solving differential equations, integral equations, and integro–differential equations (Mall and Chakraverty, 2014, 2016; Jafarian et al., 2017; Pakdaman et al., 2017; Zuniga-Aguilar et al., 2017; Rostami and Jafarian, 2018). Golbabai and Seifollahi presented radial basis function networks for solving linear Fredholm and Volterra integral equations of the second kind (Golbabai and Seifollahi, 2006), and they solved a system of nonlinear integral equations (Golbabai and Seifollahi, 2009). Effati and Buzhabadia presented multilayer perceptron networks for solving Fredholm integral equations of the second kind (Effati and Buzhabadi, 2012). Jafarian and Nia proposed a feedback neural network method for solving linear Fredholm and Volterra integral equations of the second kind (Jafarian and Nia, 2013a,b). Jafarian presented artificial neural networks-based modeling for solving the Volterra integral equations system (Jafarian et al., 2015). However, the traditional neural network algorithms have some problems, such as over-fitting, difficulty to determine hidden layer nodes, optimization of model parameters, being easily trapped into local minima, slow convergence speed, and reduction in the learning speed and efficiency of the model when the input data are large or the network structure is complex (Huang and Chen, 2008).

Huang et al. (2006a,b) proposed an extreme learning machine (ELM) algorithm, which is a single-hidden-layer feed-forward neural network. The ELM algorithm only needs to set the number of hidden nodes of the network but does not need to adjust the input weights and bias values, and the output weights can be determined by the Moore–Penrose generalized inverse operation. The ELM algorithm provides faster learning speed, better generalization performance, with least human intervention. Based on the advantages, the ELM algorithm has been widely applied to many real-world applications, such as regression and classification problems (Wong et al., 2018). Many neural network methods based on the improved extreme learning machine algorithm for solving ordinary differential equations (Yang et al., 2018; Lu et al., 2022), partial differential equations (Sun et al., 2019; Yang et al., 2020), the ruin probabilities of the classical risk model and the Erlang (2) risk model in Zhou et al. (2019), Lu et al. (2020), and one-dimensional asset-pricing (Ma et al., 2021) have been developed. Chen et al. (2020, 2021, 2022) proposed the trigonometric exponential neural network, Laguerre neural network, and neural finite element method for ruin probability, generalized Black–Scholes differential equation, and generalized Black–Scholes–Merton differential equation. Inspired by these studies, the motivation of this research is to present the sine-cosine ELM (SC-ELM) algorithm to solve linear Volterra integral equations of the first kind, linear Volterra integral equations of the second kind, linear Fredholm integral equations of the first kind, linear Fredholm integral equations of the second kind, and linear Volterra–Fredholm integral equations. In the latest study, a linear integral equation of the third kind with fixed singularities in the kernel is studied by Gabbasov and Galimova (2022), and Volterra integral equations of the first kind on a bounded interval are considered by Bulatov and Markova (2022). For more results, we may refer to Din et al. (2022) and Usta et al. (2022).

In this study, we propose a neural network method based on the sine-cosine basis function and the improved ELM algorithm to solve linear integral equations. Specifically, the hidden layer is eliminated by expanding the input pattern utilizing the sine-cosine basis function, and this simplifies the calculation to some extent. Moreover, the improved ELM algorithm can automatically satisfy the boundary conditions and it transforms the problem into solving a linear system, which provides great convenience for calculation. Furthermore, the closed-form solution by utilizing this model can be obtained, and the approximate solution of any point for linear integral equations can be provided from it.

The remainder of the article is organized as follows. In Section 2, a brief review of the ELM algorithm is provided. In Section 3, a novel neural network method based on the sine-cosine basis function and ELM algorithm for solving integral equations in the form of Equation (1) are discussed. In Section 4, we show several numerical examples to demonstrate the accuracy and the efficiency of the improved neural network algorithm. In Section 5, concluding remarks are presented.

2. The ELM algorithm

The ELM algorithm was originated from the single-hidden-layer feed-forward network (SLFN) and then got developed into a generalized SLFN algorithm (Huang and Chen, 2007). The ELM algorithm not only is fully automatically implemented without iterative tuning but also tends to the minimum training error. The ELM algorithm can provide least human intervention, faster learning speed, and better generalization performance. Therefore, the ELM algorithm is widely used in classification and regression tasks (Huang et al., 2012; Cambria and Huang, 2013).

For a data set with $N+1$ different training samples $(\boldsymbol{x_i}, \boldsymbol{g_i}) \in \mathbb{R} \times \mathbb{R}(\boldsymbol{i} = \boldsymbol{0}, \boldsymbol{1}, ..., \boldsymbol{N})$, the neural network with $M+1$ hidden neurons is expressed as follows:

$$o_i = \sum_{j=0}^{M} \beta_j f(w_j x_i + b_j), i = 0, 1, ..., N, \tag{2}$$

Where f is the activation function, w_j is the input weight of the j-th hidden layer node, b_j is the bias value of the j-th hidden layer node, and β_j is the output weight connecting the j-th hidden layer node and the output node.

The error function of SLFN is as follows:

$$e = \sum_{i=0}^{N} \|o_i - g_i\|. \tag{3}$$

Assuming the error between the output value o_i of SLFN and the exact value g_i is zero, the relationship between x_i and g_i can be modeled as follows:

$$\sum_{j=0}^{M} \beta_j f(w_j x_i + b_j) = g_i, i = 0, 1, ..., N, \tag{4}$$

Where both the input weight w_j and the bias value b_j are randomly generated. The equations (4) can be rewritten in the following matrix form, that is:

$$H\beta = G, \tag{5}$$

Where H is the output matrix of the hidden layer, and it is defined as follows:

$$H = \begin{bmatrix} f(w_0x_0+b_0) & f(w_1x_0+b_2) & \dots & f(w_Mx_0+b_M) \\ f(w_0x_1+b_1) & f(w_1x_1+b_2) & \dots & f(w_Mx_1+b_M) \\ \dots & \dots & \ddots & \dots \\ f(w_0x_N+b_1) & f(w_1x_N+b_2) & \dots & f(w_Mx_N+b_M) \end{bmatrix}_{(N+1)\times(M+1)},$$

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_N \end{bmatrix}.$$

A common minimum norm least-squares solution of the linear system Equation (5) is calculated by

$$\widehat{\beta} = \underset{\beta}{\operatorname{argmin}} \|H\beta - G\| = H^{\dagger}G. \tag{6}$$

3. The proposed method

In this section, we propose a neural network method based on sine-cosine basis function and extreme learning machine algorithm to solve linear integral equations. The single-hidden-layer sine-cosine neural network algorithm consists of three layers: an input layer, a hidden layer, and an output layer. The unique hidden layer consists of two parts. The first part uses the cosine basis function as the basis function and the other part implements the superposition of the sine basis function. The structure of sine-cosine neural network method is shown in Figure 1.

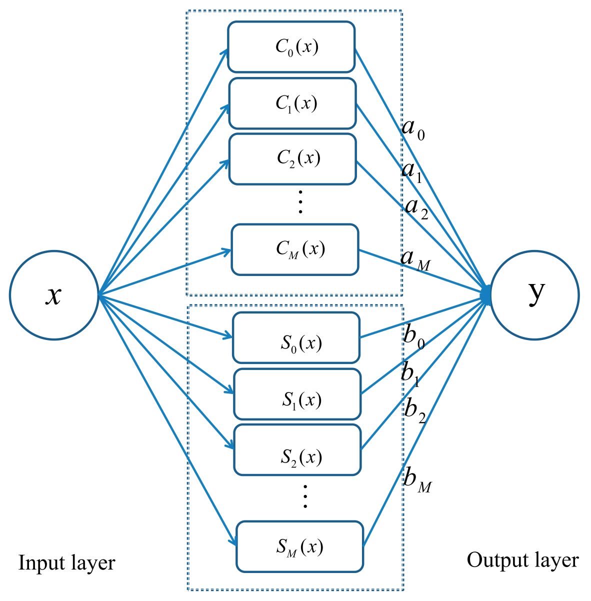

FIGURE 1
The structure of sine-cosine neural network method for solving several kinds of linear integral equations.

The steps of the sine-cosine neural network method for solving several kinds of linear integral equations are as follows:

Step 1: Discretize the interval $[a, b]$ into a series of collocation points $\Omega = \{a = x_0 < x_1 < ... < x_N = b\}$, $x_i = a + \frac{b-a}{N} i, i = 0, 1, ..., N$.

Step 2: Construct the approximate solution by using sine-cosine basis as an activation function, that is

$$\hat{y}_{\text{SC-ELM}}(x) = \sum_{j=0}^{M} a_j cos\left(\frac{j\pi}{b-a}(x-a)\right) + \sum_{j=0}^{M} b_j sin\left(\frac{j\pi}{b-a}(x-a)\right). \tag{7}$$

Step 3: According to different problems and different data sets, we substitute the trial solution $\hat{y}_{\text{SC-ELM}}$ into the Equation (1) to be solved. Then, we convert this equation into a matrix form:

$$\|H\widehat{\beta} - G\| = \min_{\beta} \|H\beta - G\|, \tag{8}$$

Where $H = \{[ha_{ij}]_{N+1,M+1}, [hb_{ij}]_{N+1,M+1}\}$; $\beta = (a_0, a_1, ..., a_M, b_0, b_1, ..., b_M)'$;
$ha_{ij} = \epsilon cos\left(\frac{j\pi}{b-a}(x_i - a)\right) + \lambda \int_a^b k_1(x_i, t) cos\left(\frac{j\pi}{b-a}(t-a)\right) dt + \mu \int_a^{x_i} k_2(x_i, t) cos\left(\frac{j\pi}{b-a}(t-a)\right) dt, i = 0, 1, ..., N, j = 0, 1, ..., M$;
$hb_{ij} = \epsilon sin\left(\frac{j\pi}{b-a}(x_i - a)\right) + \lambda \int_a^b k_1(x_i, t) sin\left(\frac{j\pi}{b-a}(t-a)\right) dt + \mu \int_a^{x_i} k_2(x_i, t) sin\left(\frac{j\pi}{b-a}(t-a)\right) dt, i = 0, 1, ..., N, j = 0, 1, ..., M$;
$G = (g(x_0), g(x_1), ..., g(x_N))'$.

Step 4: From the theory of Moore–Penrose generalized inverse of matrix H, we can obtain the net parameters as

$$\widehat{\beta} = H^{\dagger} G = \underset{\beta}{\text{argmin}} \|H\beta - G\|. \tag{9}$$

Step 5: Find the connection parameters a_j, b_j and the number of neurons M with the smallest MSE as the optimal value. The corresponding optimal number of neurons M and output weights a_j, b_j are, respectively, the optimal number of neurons M and optimal output weights $\widehat{\beta}$.

Step 6: Substitute $a_j, b_j, j = 0, 1, 2, \ldots, M$ into Equation (7) to get the new numerical solution.

Some advantages of the single-layer sine-cosine neural network method for solving integral equations are as follows:

(i) The hidden layer is eliminated by expanding the input pattern using the sine-cosine basis function.
(ii) The sine-cosine neural network algorithm only needs to determine the weights of the output layer. The problem could be transformed into a linear system, and the output weights can be obtained by a simple generalized inverse matrix, which greatly improves the calculation speed.
(iii) We can obtain the closed-form solution by using this model, and most important of all, the approximate solution of any

TABLE 1 Comparison between the exact solution and approximate solution (Example 1).

x	Exact solution	Approximate solution	Absolute error
0.0624	0.99805375164	0.99805375126	3.8610e-10
0.0915	0.99581679479	0.99581679410	6.9084e-10
0.1518	0.98850048763	0.98850048732	3.1110e-10
0.2410	0.97109978660	0.97109978647	1.2768e-10
0.3604	0.93575583912	0.93575583904	7.4031e-11
0.5252	0.86522368172	0.86522368169	2.7006e-11
0.6395	0.80239425533	0.80239425500	3.2675e-10
0.7590	0.72552456965	0.72552456924	4.1422e-10
0.8482	0.66133438071	0.66133438024	4.7606e-10
0.9084	0.61500816934	0.61500816901	3.3012e-10
0.9348	0.59397933431	0.59397933361	6.9796e-10

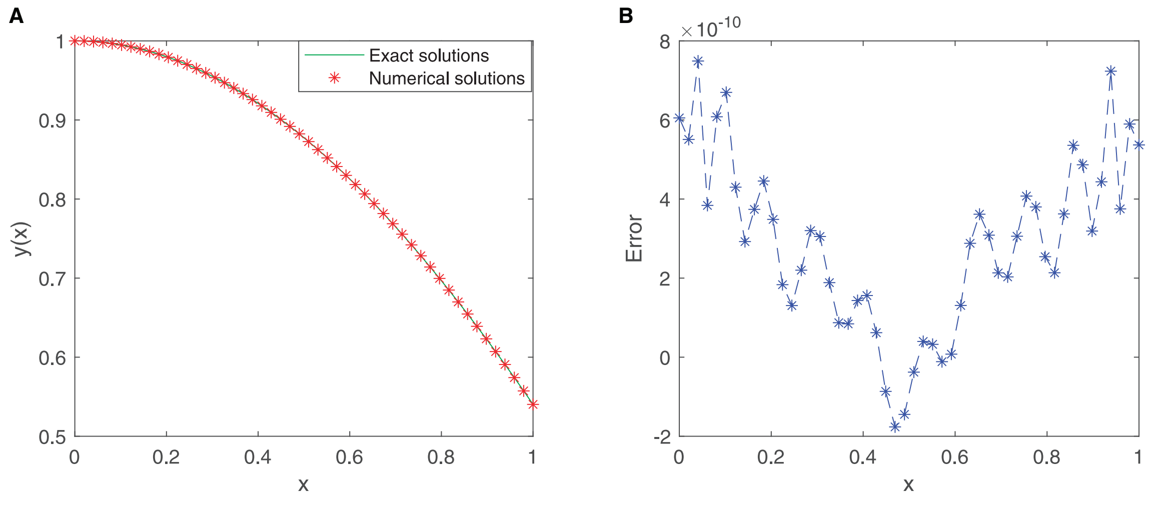

FIGURE 2
(A) Comparison between exact and SC-ELM solutions for Example 1. **(B)** Errors of Example 1.

point for linear integral equations can be given from it. It provides a good method for solving integral equations.

4. Numerical experiments

In this section, some numerical experiments are performed to demonstrate the reliability and powerfulness of the improved neural network algorithm. The sine-cosine neural network method based on the sine-cosine basis function and ELM algorithm is applied to solve the linear Volterra integral equations of the first kind, linear Volterra integral equations of the second kind, linear Fredholm integral equations of the first kind, linear Fredholm integral equations of the second kind, and linear Volterra–Fredholm integral equations.

The algorithm is evaluated with MATLAB R2021a running in an Intel Xeon Gold 6226R CPU with 64.0GB RAM. The training set is obtained by taking points at equal intervals, and the testing set is randomly selected. The validation set is the set of midpoints $V = \{v_i | v_i = (x_i + x_{i+1})/2, i = 0, 1, ..., N\}$, where $\{x_i\}_{i=0}^N$ are training points in the following studies. We use mean square error (MSE), absolute error (AE), mean absolute error (MAE) and root mean square error (RMSE) to measure the error of numerical solution. They can be defined as follows:

$$\begin{aligned} MSE &= \frac{1}{N+1}\sum_{i=0}^{N}(y(x_i) - \hat{y}(x_i))^2, \\ AE &= |y(x_i) - \hat{y}(x_i)|, \\ RMSE &= \left[\frac{1}{N+1}\sum_{i=0}^{N}(y(x_i) - \hat{y}(x_i))^2\right]^{\frac{1}{2}}, \end{aligned} \quad (10)$$

Where $y(x_i)$ denote the exact solution and $\hat{y}(x_i)$ represent the approximate solution obtained by the proposed algorithm. Note

TABLE 2 Comparison between the SC-ELM method and the LS-SVR method (Example 2).

x	LS-SVR in Guo et al. (2012)	SC-ELM
0.1	7.4597e-08	6.8246e-10
0.2	2.7590e-08	3.7957e-10
0.3	5.1917e-09	3.2404e-10
0.4	2.3898e-07	1.6271e-10
0.5	2.4981e-07	9.4236e-11
0.6	3.8031e-08	4.6072e-11
0.7	2.3423e-07	1.9703e-10
0.8	5.2083e-08	2.3283e-10
0.9	2.4366e-07	3.1284e-10

TABLE 3 Comparison between exact solution and approximate solution (Example 2).

x	Exact solution	Approximate solution	Absolute error
0.0624	0.93950700882	0.93950703293	2.4118e-08
0.0915	0.91256131615	0.91256129442	2.1728e-08
0.1518	0.85916009558	0.85916009819	2.6073e-09
0.2410	0.78584162639	0.78584161623	1.0154e-08
0.3604	0.69739731135	0.69739729287	1.8481e-08
0.5252	0.59143706512	0.59143704802	1.7099e-08
0.6395	0.52755613618	0.52755611864	1.7535e-08
0.7590	0.46813432735	0.46813431582	1.1524e-08
0.8482	0.42818497165	0.42818496663	5.0275e-09
0.9084	0.40316877830	0.40316880212	2.3823e-08
0.9348	0.39266439056	0.39266439284	2.2803e-09

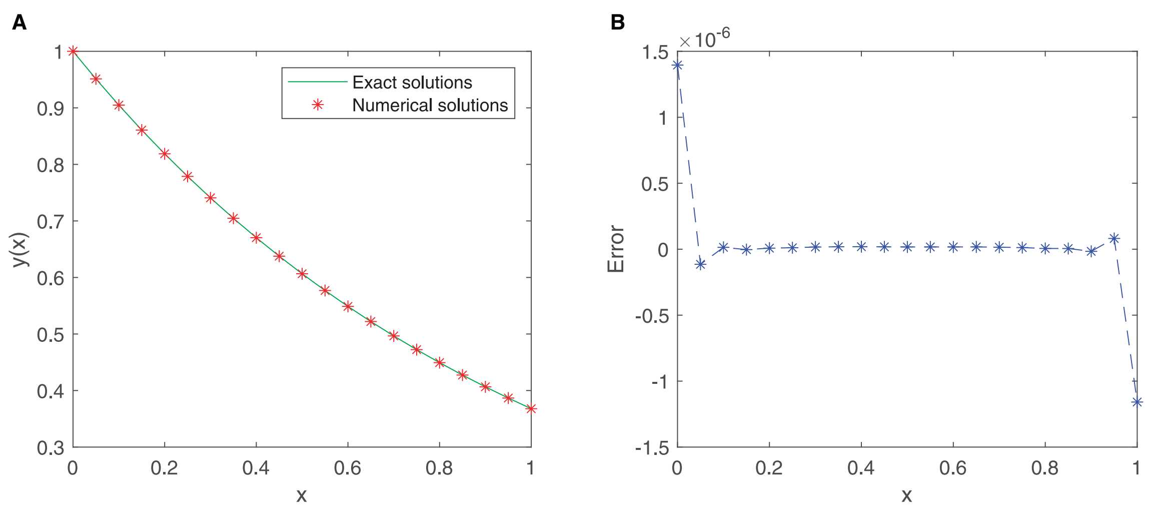

FIGURE 3
(A) Comparison between exact and SC-ELM solutions for Example 2. **(B)** Errors of Example 2.

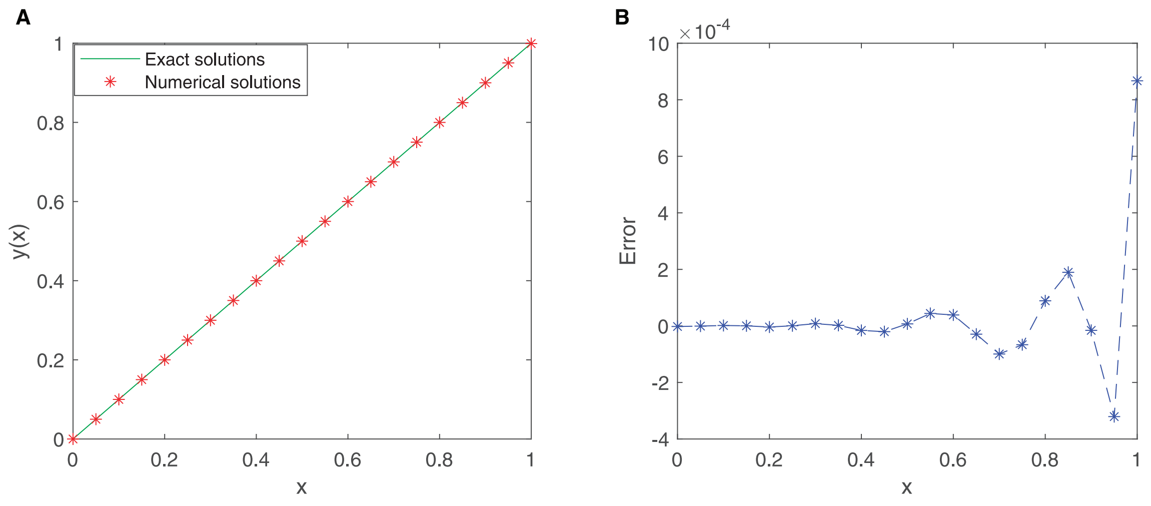

FIGURE 4
(A) Comparison between exact and SC-ELM solutions for Example 3. **(B)** Errors of Example 3.

that $w_j = j\pi/(b-a)$ and $b_j = -j\pi a/(b-a)(j = 0, 1, 2, ..., M)$ are selected in our proposed method. Moreover, the number M of hidden neurons that results in minimum mean squared error on the validation set can be selected.

4.1. Example 1

Consider linear Volterra integral equation of the second kind (Guo et al., 2012) as

$$f(x) + \int_0^x (x-t)f(t)dt = 1, \quad x \in [0,1], \tag{11}$$

The analytical solution is $f(x) = cos(x)$.

We train our proposed neural network for 50 equidistant points in the given interval $[0, 1]$ with the first 12 sine-cosine basis functions. Comparison between the exact solution and the approximate solution *via* our improved neural network algorithm is depicted in Figure 2A, and the plot of the error function between them is cited in Figure 2B. As shown in the figures, the mean squared error is 1.3399×10^{-19}, and the maximum absolute error is approximately 7.4910×10^{-10}.

Table 1 incorporates the results of the exact solution and the approximate solution *via* our proposed neural network algorithm for 11 testing points at unequal intervals in the domain $[0, 1]$. The absolute errors are listed in Table 1, in which we observe that the mean squared error is approximately 1.6789×10^{-19}. These results imply that the proposed method has higher accuracy.

Table 2 compares the proposed method with the LS-SVR method. The maximum absolute error is approximately 6.8246×10^{-10}. Note that in Guo et al. (2012), the maximum absolute error shown in Guo et al. (2012) Table 5 is approximately 2.4981×10^{-7}. The solution accuracy of the proposed algorithm is higher.

TABLE 4 Comparison between the exact solution and approximate solution (Example 3).

x	Exact solution	Approximate solution	Absolute error
0.0624	0.0624	0.0624007691887	7.6919e-07
0.0915	0.0915	0.0914990435138	9.5649e-07
0.1518	0.1518	0.1517998073669	1.9263e-07
0.2410	0.2410	0.2410008795702	8.7957e-07
0.3604	0.3604	0.3604013108547	1.3109e-06
0.5252	0.5252	0.5251722711564	2.7729e-05
0.6395	0.6395	0.6395114159462	1.1416e-05
0.7590	0.7590	0.7590451800564	4.5180e-05
0.8482	0.8482	0.8480097658921	1.9023e-04
0.9084	0.9084	0.9084802272880	8.0227e-05
0.9348	0.9348	0.9350743334136	2.7433e-04

4.2. Example 2

Consider the linear Volterra integral equation of the first kind (Masouri et al., 2010) as

$$\int_0^x e^{x+t} f(t)dt = xe^x, \quad x \in [0,1]. \tag{12}$$

The analytical solution is $f(x) = e^{-x}$.

A total of 21 equidistant points in the given interval $[0, 1]$ are used as the training points, and the neural network adapts the first 10 sine-cosine basis functions. Figures 3A, B shows that the exact solution and the approximate solution are highly consistent. The maximum absolute error is approximately 1.3959×10^{-6}.

Table 3 lists the results of the exact solution and the approximate solution *via* our proposed neural network algorithm in the domain $[0, 1]$. The mean squared error is approximately

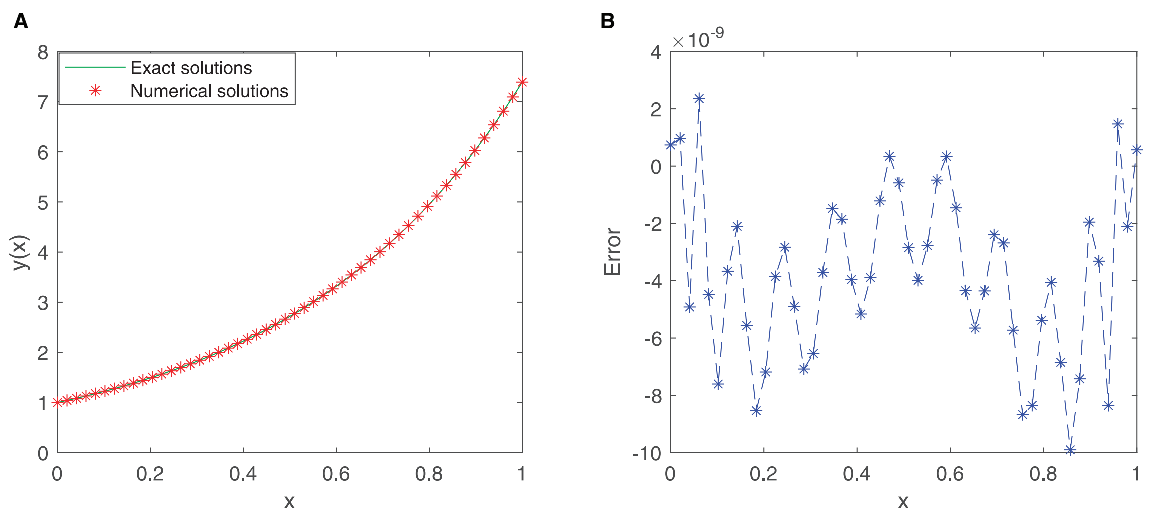

FIGURE 5
(A) Comparison between exact and SC-ELM solutions for Example 4. **(B)** Errors of Example 4.

2.5781×10^{-16}. These findings provide a strong support for the effectiveness of our proposed method.

4.3. Example 3

We consider linear Fredholm integral equation of the first kind (Rashed, 2003) as

$$\int_0^1 (x^2 + t^2)^{\frac{1}{2}} f(t)dt = \frac{(1 + x^2)^{\frac{3}{2}} - x^3}{3}. \tag{13}$$

The analytical solution is $f(x) = x$.

This problem is solved by utilizing our proposed neural network model in the given interval $[0, 1]$. We consider 21 equidistant points in the domain $[0, 1]$ with the first six sine-cosine basis functions to train the model. Comparison between the exact solution and the approximate solution *via* our improved neural network algorithm is depicted in Figure 4A, and the error plot is depicted in Figure 4B. Note that the mean squared error is 4.5915×10^{-8} for these training points.

Table 4 incorporates the results of the exact solution and the approximate solution *via* our proposed neural network algorithm for 11 testing points at unequal intervals in the domain $[0, 1]$. We observe that the maximum absolute error is approximately 2.7433×10^{-4}. The results show that this new neural network has a good generalization ability.

4.4. Example 4

We consider the linear Fredholm integral equation of the second kind (Golbabai and Seifollahi, 2006) as

$$f(x) + \frac{1}{3}\int_0^1 e^{2x - \frac{5t}{3}} f(t)dt = e^{2x + \frac{1}{3}}. \tag{14}$$

The analytical solution is $f(x) = e^{2x}$.

TABLE 5 Comparison between the exact solution and approximate solution (Example 4).

x	Exact solution	Approximate solution	Absolute error
0.0624	1.13292184603767	1.13292184381566	2.2220e−09
0.0915	1.20081440808083	1.20081441530638	7.2255e−09
0.1518	1.35472705687431	1.35472706011551	3.2412e−09
0.2410	1.61930978530193	1.61930978804438	2.7425e−09
0.3604	2.05607741480638	2.05607741623227	1.4259e−09
0.5252	2.85879440715296	2.85879441106042	3.9075e−09
0.6395	3.59304488356426	3.59304488863783	5.0735e−09
0.7590	4.56308988310901	4.56308989202265	8.9136e−09
0.8482	5.45427660976895	5.45427661875143	8.9825e−09
0.9084	6.15214006907956	6.15214007048970	1.4101e−09
0.9348	6.48570159972151	6.48570160778124	8.0597e−09

The improved neural network algorithm for the linear Fredholm integral equation of the second kind has been trained with 50 equidistant points in the given interval $[0, 1]$ with the first 12 sine-cosine basis functions. The approximate solution obtained by the improved neural network algorithm and the exact solution are shown in Figure 5A, and the error function is displayed in Figure 5B. Especially, the mean squared error is 2.3111×10^{-17}, and the maximum absolute error is approximately 9.8998×10^{-9}, which fully demonstrates the superiority of the improved neural network algorithm.

Finally, Table 5 provides the results of the exact solution and the approximate solution *via* our proposed neural network algorithm for 11 testing points at unequal intervals in the domain $[0, 1]$. As shown in Table 5, the mean squared error is approximately $3.1391 \times$

10^{-17}, which undoubtedly shows the power and effectiveness of the proposed method.

Table 6 compares the proposed method with RBF networks. The maxmium absolute error by our proposed method is approximately 7.7601×10^{-9}. Note that in Golbabai and Seifollahi (2006), the maxmium absolute error shown in Golbabai and Seifollahi (2006), as shown in Table 1, is approximately 6.7698×10^{-7}. The solution accuracy of the proposed algorithm is higher.

4.5. Example 5

Consider the linear Volterra–Fredholm integral equation (Wang and Wang, 2014) as

$$y(x) + \int_0^1 e^{x+t} y(t)dt - \int_0^x e^{x+t} y(t)dt = e^{-x} - e^x(x-1). \quad (15)$$

The analytical solution is $f(x) = e^{-x}$.

TABLE 6 Comparison between the SC-ELM method and RBF method (Example 4).

x	RBF in Golbabai and Seifollahi (2006)	SC-ELM
0.1	4.1721e-07	7.7331e-09
0.2	1.6226e-07	7.7601e-09
0.3	9.9728e-08	7.0314e-09
0.4	5.3328e-07	4.9446e-09
0.5	5.1282e-07	1.7010e-09
0.6	8.8658e-08	9.5548e-11
0.7	3.8239e-07	2.1508e-09
0.8	6.7698e-07	4.8329e-09
0.9	3.3687e-07	1.6513e-09

A total of 50 equidistant points in the given interval [0, 1] and the first 11 sine-cosine basis functions are considered to train the neural network model. The comparison images and error images of the exact solution and the approximate solution are listed in Figures 6A, B, from which we can see that the mean squared error is 3.3499×10^{-18}.

Table 7 shows the results of the exact solution and the approximate solution *via* the improved ELM method for 11 testing points at unequal intervals in the domain [0, 1]. As shown in the table, the maximum absolute error is approximately 2.6673×10^{-9}, which reveals that the improved neural network algorithm has higher accuracy and excellent performance.

We compare the RMSE of our proposed method and the Taylor collocation method in Wang and Wang (2014). From Table 8, we can see clearly that our algorithm is more accurate than the

TABLE 7 Comparison between the exact solution and approximate solution (Example 5).

x	Exact solution	Approximate solution	Absolute error
0.0624	0.93950700882	0.93950700692	1.8957e-09
0.0915	0.91256131615	0.91256131348	2.6673e-09
0.1518	0.85916009558	0.85916009332	2.2550e-09
0.2410	0.78584162639	0.78584162475	1.6352e-09
0.3604	0.69739731135	0.69739731026	1.0886e-09
0.5252	0.59143706512	0.59143706314	1.9856e-09
0.6395	0.52755613618	0.52755613446	1.7191e-09
0.7590	0.46813432735	0.46813432542	1.9239e-09
0.8482	0.42818497165	0.42818496954	2.1172e-09
0.9084	0.40316877830	0.40316877665	1.6478e-09
0.9348	0.39266439056	0.39266438843	2.1275e-09

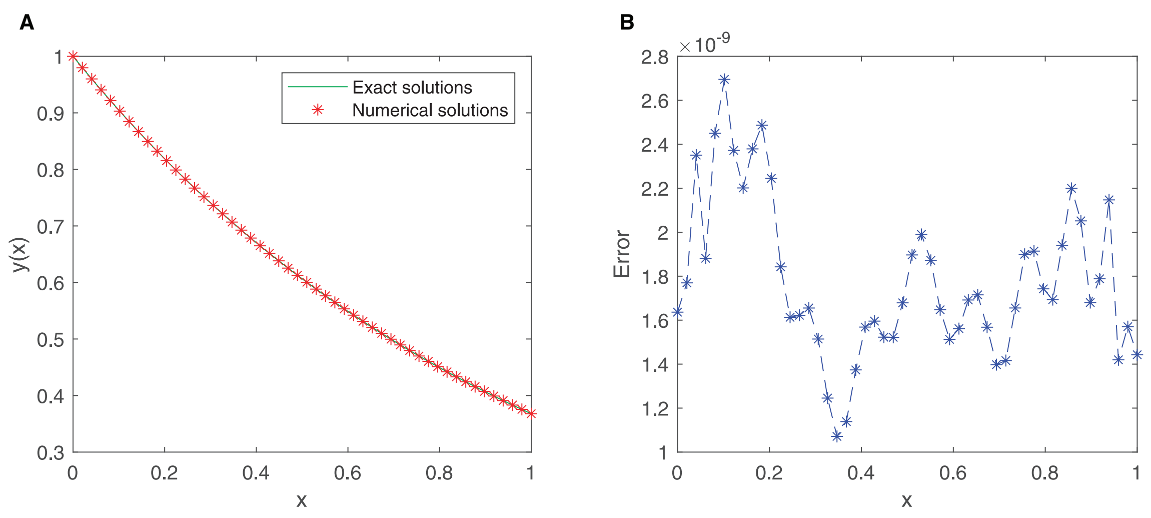

FIGURE 6
(A) Comparison between exact and SC-ELM solutions for Example 5. **(B)** Errors of Example 5.

algorithm in the Taylor collocation method. As can be seen from Table 8, when 5, 8, and 9 points are tested, the RMSEs shown by the Taylor collocation method in Wang and Wang (2014) are approximately 4.03×10^{-7}, 9.50×10^{-7}, and 2.15×10^{-5}, but the RMSEs shown by our proposed method are respectively 1.67×10^{-9}, 1.78×10^{-9}, and 1.67×10^{-9}.

4.6. Example 6

We consider linear the Volterra integral equation of the second kind (Saberi-Nadjafi et al., 2012).

$$f(x) + \int_0^x (8-2x^2)sin(xt)f(t)dt = -2x + 4sin(x^2)(sin2x - cos2x) + (sin2x + cos2x)(1 + 2xcos(x^2)). \quad (16)$$

The analytical solution is $f(x) = sin(2x) + cos(2x)$.

A total of 21 equidistant discrete points and the first 11 sine-cosine basis functions are utilized to construct the neural network model. The comparison images and error images of the exact solution and the approximate solution are displayed in Figures 7A, B. It ia not hard to find that the MSE is 4.2000×10^{-16}, and this implies that the proposed algorithm has higher accuracy.

To verify the effectiveness of our proposed method, we provide the results of the exact solution and the approximate solution *via* the improved ELM method for 11 testing points at unequal intervals in the domain $[0, 1]$, see Table 9. As shown in the table, the maximum absolute error is approximately 2.9539×10^{-8}, which shows that the proposed algorithm has good generalization ability.

TABLE 8 RMSE comparison of Example 5.

N	Taylor solution	SC-ELM solution
5	4.03e-07	1.67e-09
8	9.50e-07	1.78e-09
9	2.15e-05	1.67e-09

TABLE 9 Comparison between the exact solution and approximate solution (Example 6).

x	Exact solution	Approximate solution	Absolute error
0.0624	1.1166988736914	1.1166988834032	9.7117e-09
0.0915	1.1652824720246	1.1652824868524	1.4828e-08
0.1518	1.2532239237305	1.2532239421393	1.8409e-08
0.2410	1.3496218317010	1.3496218507822	1.9081e-08
0.3604	1.4112638863693	1.4112639090979	2.2729e-08
0.5252	1.3648462234191	1.3648462529585	2.9539e-08
0.6395	1.2454017480945	1.2454017691674	2.1073e-08
0.7590	1.0513783999948	1.0513784162122	1.6217e-08
0.8482	0.8668485498714	0.8668485662364	1.6365e-08
0.9084	0.7263634857982	0.7263635021492	1.6351e-08
0.9348	0.6613122433253	0.6613122627673	1.9442e-08

TABLE 10 Comparison of the different examples of MSE with different numbers of training points and hidden neurons.

MSE	$M = 5$, $N = 20$	$M = 10$, $N = 20$	$M = 10$, $N = 100$
Example 1	5.4420e-11	3.6233e-17	6.9420e-19
Example 2	2.7384e-09	1.9056e-12	6.3433e-17
Example 3	4.5915e-08	4.4380e-05	4.8435e-06
Example 4	2.8418e-08	4.5969e-16	3.8451e-16
Example 5	1.6398e-10	8.0501e-17	2.0969e-18
Example 6	2.8025e-11	4.2000e-16	4.0548e-19

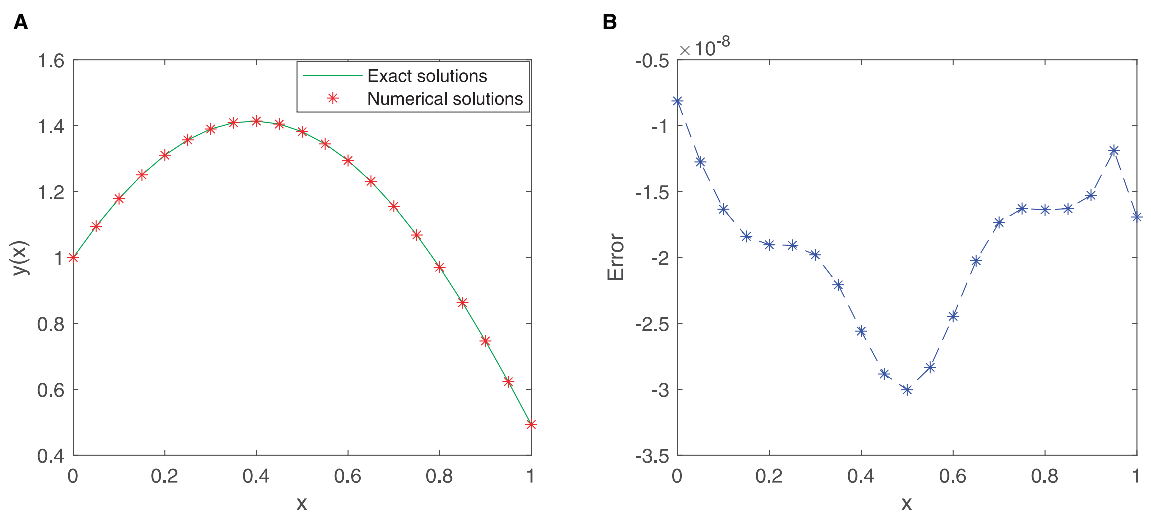

FIGURE 7
(A) Comparison between exact and SC-ELM solutions for Example 6. **(B)** Errors of Example 6.

TABLE 11 Execution time of different examples.

Example	*t*
Example 1	0.3317
Example 2	0.1505
Example 3	0.1080
Example 4	0.3391
Example 5	0.6356
Example 6	0.1683

Table 10 compares the MSE of the numerical solutions obtained by the SC-ELM model when more training points are added and different numbers of hidden layer neurons are configured. From these results, it can be seen that the proposed method can achieve good accuracy. The calculation time of different examples is listed in Table 11. These data suggest that our method is efficient and feasible.

5. Conclusion

In this study, the improved neural network algorithm based on the sine-cosine basis function and extreme learning machine algorithm has been developed for solving linear integral equations. The accuracy of the improved neural network has been checked by solving a linear Volterra integral equation of the first kind, a linear Volterra integral equation of the second kind, a linear Fredholm integral equation of the first kind, a linear Fredholm integral equation of the second kind, and a linear Volterra-Fredholm integral equation. The experimental results of the improved ELM approach with different types of integral equations show that the simulation results are close to the exact results. Therefore, the proposed model is very precise and could be a good tool for solving linear integral equations.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

Acknowledgments

The authors sincerely thank all the reviewers and the editor for their careful reading and valuable comments, which improved the quality of this study.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abdou, M. A. (2002). Fredholm-Volterra integral equation of the first kind and contact problem. *Appl. Math. Comput.* 125, 177–193. doi: 10.1016/S0096-3003(00)00118-1

Bulatov, M. V., and Markova, E. V. (2022). Collocation-variational approaches to the solution to volterra integral equations of the first kind. *Comput. Math. Math. Phys.* 62, 98–105. doi: 10.1134/S0965542522010055

Cambria, E., and Huang, G. B. (2013). Extreme learning machine: trends and controversies. *IEEE Intell. Syst.* 28, 30–59. doi: 10.1109/MIS.2013.140

Chen, Y., Wei, L., Cao, S., Liu, F., Yang, Y., and Cheng, Y. (2022). Numerical solving for generalized Black-Scholes-Merton model with neural finite element method. *Digit. Signal Process.* 131, 103757. doi: 10.1016/j.dsp.2022.103757

Chen, Y., Yi, C., Xie, X., Hou, M., and Cheng, Y. (2020). Solution of ruin probability for continuous time model based on block trigonometric exponential neural network. *Symmetry* 12, 876. doi: 10.3390/sym12060876

Chen, Y., Yu, H., Meng, X., Xie, X., Hou, M., and Chevallier, J. (2021). Numerical solving of the generalized Black-Scholes differential equation using Laguerre neural network. *Digit. Signal Process.* 112, 103003. doi: 10.1016/j.dsp.2021.103003

Dastjerdi, H. L., and Ghaini, F. M. M. (2012). Numerical solution of Volterra-Fredholm integral equations by moving least square method and Chebyshev polynomials. *Appl. Math. Model* 36, 3283–3288. doi: 10.1016/j.apm.2011.10.005

Din, Z. U., Islam, S. U., and Zaman, S. (2022). Meshless procedure for highly oscillatory kernel based one-dimensional volterra integral equations. *J. Comput. Appl. Math.* 413, 114360. doi: 10.1016/j.cam.2022.114360

Effati, S., and Buzhabadi, R. (2012). A neural network approach for solving Fredholm integral equations of the second kind. *Neural Comput. Appl.* 21, 843–852. doi: 10.1007/s00521-010-0489-y

Gabbasov, N. S., and Galimova, Z. K. (2022). On numerical solution of one class of integral equations of the third kind. *Comput. Math. Math. Phys.* 62, 316–324. doi: 10.1134/S0965542522020075

Golbabai, A., and Seifollahi, S. (2006). Numerical solution of the second kind integral equations using radial basis function networks. *Appl. Math. Comput.* 174, 877–883. doi: 10.1016/j.amc.2005.05.034

Golbabai, A., and Seifollahi, S. (2009). Solving a system of nonlinear integral equations by an RBF network. *Comput. Math. Appl.* 57, 1651–1658. doi: 10.1016/j.camwa.2009.03.038

Guo, X. C., Wu, C. G., Marchese, M., and Liang, Y. C. (2012). LS-SVR-based solving Volterra integral equations. *Appl. Math. Comput.* 218, 11404–11409. doi: 10.1016/j.amc.2012.05.028

Habib, G., and Qureshi, S. (2022). Global Average Pooling convolutional neural network with novel NNLU activation function and HYBRID parallelism. *Front. Comput. Neurosci.* 16, 1004988. doi: 10.3389/fncom.2022.1004988

Hou, M., and Han, X. (2012). Multivariate numerical approximation using constructive $L^2(R)$ RBF neural network. *Neural Comput. Appl.* 21, 25–34. doi: 10.1007/s00521-011-0604-8

Hou, M., Liu, T., Yang, Y., Zhu, H., Liu, H., Yuan, X., et al. (2017). A new hybrid constructive neural network method for impacting and its application on tungsten rice prediction. *Appl. Intell.* 47, 28–43. doi: 10.1007/s10489-016-0882-z

Hou, M., Yang, Y., Liu, T., and Peng, W. (2018). Forecasting time series with optimal neural networks using multi-objective optimization algorithm based on AICc. *Front. Comput. Sci.* 12, 1261–1263. doi: 10.1007/s11704-018-8095-8

Huang, G. B., and Chen, L. (2007). Letters: Convex incremental extreme learning machine. *Neurocomputing* 70, 3056–3062. doi: 10.1016/j.neucom.2007.02.009

Huang, G. B., and Chen, L. (2008). Enhanced random search based incremental extreme learning machine. *Neurocomputing* 71, 3460–3468. doi: 10.1016/j.neucom.2007.10.008

Huang, G. B., Chen, L., and Siew, C. K. (2006a). Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Netw.* 17, 879–892. doi: 10.1109/TNN.2006.875977

Huang, G. B., Zhou, H., Ding, X., and Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Trans. Syst. Man Cybern. B Cybern.* 42, 513–529. doi: 10.1109/TSMCB.2011.2168604

Huang, G. B., Zhu, Q. Y., and Siew, C. K. (2006b). Extreme learning machine: theory and applications. *Neurocomputing* 70, 489–501. doi: 10.1016/j.neucom.2005.12.126

Isaacson, S. A., and Kirby, R. M. (2011). Numerical solution of linear Volterra integral equations of the second kind with sharp gradients. *J. Comput. Appl. Math.* 235, 4283–4301. doi: 10.1016/j.cam.2011.03.029

Jafarian, A., Measoomy, S., and Abbasbandy, S. (2015). Artificial neural networks based modeling for solving Volterra integral equations system. *Appl. Soft. Comput.* 27, 391–398. doi: 10.1016/j.asoc.2014.10.036

Jafarian, A., Mokhtarpour, M., and Baleanu, D. (2017). Artificial neural network approach for a class of fractional ordinary differential equation. *Neural Comput. Appl.* 28, 765–773. doi: 10.1007/s00521-015-2104-8

Jafarian, A., and Nia, S. M. (2013a). Feedback neural network method for solving linear Volterra integral equations of the second kind. *Int. J. Math. Model. Numer. Optim.* 4, 225–237. doi: 10.1504/IJMMNO.2013.056531

Jafarian, A., and Nia, S. M. (2013b). Using feed-back neural network method for solving linear Fredholm integral equations of the second kind. *J. Hyperstruct* 2, 53–71.

Li, Y. F., and Ying, H. (2022). Disrupted visual input unveils the computational details of artificial neural networks for face perception. *Front. Comput. Neurosci.* 16, 1054421. doi: 10.3389/fncom.2022.1054421

Lima, P. M., and Buckwar, E. (2015). Numerical solution of the neural field equation in the two-dimensional case. *SIAM J. Sci. Comput.* 37, B962-B979. doi: 10.1137/15M1022562

Lu, Y., Chen, G., Yin, Q., Sun, H., and Hou, M. (2020). Solving the ruin probabilities of some risk models with Legendre neural network algorithm. *Digit. Signal Process.* 99, 102634. doi: 10.1016/j.dsp.2019.102634

Lu, Y., Weng, F., and Sun, H. (2022). Numerical solution for high-order ordinary differential equations using H-ELM algorithm. *Eng. Comput.* 39, 2781–2801. doi: 10.1108/EC-11-2021-0683

Ma, M., Zheng, L., and Yang, J. (2021). A novel improved trigonometric neural network algorithm for solving price-dividend functions of continuous time one-dimensional asset-pricing models. *Neurocomputing* 435, 151–161. doi: 10.1016/j.neucom.2021.01.012

Maleknejad, K., and Mirzaee, F. (2005). Using rationalized Haar wavelet for solving linear integral equations. *Appl. Math. Comput.* 160, 579–587. doi: 10.1016/j.amc.2003.11.036

Mall, S., and Chakraverty, S. (2014). Chebyshev neural network based model for solving Lane-Emden type equations. *Appl. Math. Comput.* 247, 100–114. doi: 10.1016/j.amc.2014.08.085

Mall, S., and Chakraverty, S. (2016). Application of Legendre neural network for solving ordinary differential equations. *Appl. Soft. Comput.* 43, 347–356. doi: 10.1016/j.asoc.2015.10.069

Mandal, B. N., and Bhattacharya, S. (2007). Numerical solution of some classes of integral equations using Bernstein polynomials. *Appl. Math. Comput.* 190, 1707–1716. doi: 10.1016/j.amc.2007.02.058

Masouri, Z., Babolian, E., and Hatamzadeh-Varmazyar, S. (2010). An expansion-iterative method for numerically solving Volterra integral equation of the first kind. *Comput. Math. Appl.* 59, 1491–1499. doi: 10.1016/j.camwa.2009.11.004

Mirzaee, F., and Hoseini, S. F. (2016). Application of Fibonacci collocation method for solving Volterra-Fredholm integral equations. *Appl. Math. Comput.* 273, 637–644. doi: 10.1016/j.amc.2015.10.035

Nemati, S. (2015). Numerical solution of Volterra-Fredholm integral equations using Legendre collocation method. *J. Comput. Appl. Math.* 278, 29–36. doi: 10.1016/j.cam.2014.09.030

Pakdaman, M., Ahmadian, A., Effati, S., Salahshour, S., and Baleanu, D. (2017). Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Appl. Math. Comput.* 293, 81–95. doi: 10.1016/j.amc.2016.07.021

Rashed, M. T. (2003). Numerical solution of the integral equations of the first kind. *Appl. Math. Comput.* 145, 413–420. doi: 10.1016/S0096-3003(02)00497-6

Rashidinia, J., and Zarebnia, M. (2007). Solution of Voltera integral equation by the Sinc-collection method. *J. Comput. Appl. Math.* 206, 801–813. doi: 10.1016/j.cam.2006.08.036

Rostami, F., and Jafarian, A. (2018). A new artificial neural network structure for solving high-order linear fractional differential equations. *Int. J. Comput. Math.* 95, 528–539. doi: 10.1080/00207160.2017.1291932

Saberi-Nadjafi, J., Mehrabinezhad, M., and Akbari, H. (2012). Solving Volterra integral equations of the second kind by wavelet-Galerkin scheme. *Comput. Math. Appl.* 63, 1536–1547. doi: 10.1016/j.camwa.2012.03.043

Sun, H., Hou, M., Yang, Y., Zhang, T., Weng, F., and Han, F. (2019). Solving partial differential equation based on bernstein neural network and extreme learning machine algorithm. *Neural Process. Lett.* 50, 1153–1172. doi: 10.1007/s11063-018-9911-8

Usta, F., Akyiğit, M., Say, F., and Ansari, K. J. (2022). Bernstein operator method for approximate solution of singularly perturbed volterra integral equations. *J. Math. Anal. Appl.* 507, 125828. doi: 10.1016/j.jmaa.2021.125828

Wang, K. Y., and Wang, Q. S. (2013). Lagrange collocation method for solving Volterra-Fredholm integral equations. *Appl. Math. Comput.* 219, 10434–10440. doi: 10.1016/j.amc.2013.04.017

Wang, K. Y., and Wang, Q. S. (2014). Taylor collocation method and convergence analysis for the Volterra-Fredholm integral equations. *J. Comput. Appl. Math.* 260, 294–300. doi: 10.1016/j.cam.2013.09.050

Wong, C., Vong, C., Wong, P., and Cao, J. (2018). Kernel-based multilayer extreme learning machines for representation learning. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 757–762. doi: 10.1109/TNNLS.2016.2636834

Yang, Y., Hou, M., and Luo, J. (2018). A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods. *Adv. Diff. Equat.* 469, 1–24. doi: 10.1186/s13662-018-1927-x

Yang, Y., Hou, M., Sun, H., Zhang, T., Weng, F., and Luo, J. (2020). Neural network algorithm based on Legendre improved extreme learning machine for solving elliptic partial differential equations. *Soft Comput.* 24, 1083–1096. doi: 10.1007/s00500-019-03944-1

Zhou, T., Liu, X., Hou, M., and Liu, C. (2019). Numerical solution for ruin probability of continuous time model based on neural network algorithm. *Neurocomputing* 331, 67–76. doi: 10.1016/j.neucom.2018.08.020

Zuniga-Aguilar, C. J., Romero-Ugalde, H. M., Gomez-Aguilar, J. F., Jimenez, R. F. E., and Valtierra, M. (2017). Solving fractional differential equations of variable-order involving operators with Mittag-Leffler kernel using artificial neural networks. *Chaos Solitons Fractals* 103, 382–403. doi: 10.1016/j.chaos.2017.06.030